Final Report

Title:
Timed Formalisms for Plan Ontology and Processes

Principal Investigator:
Dr. Jin Song Dong,
School of Computing, National University of Singapore

Researchers:
Dr. Jun Sun (Postdoc), Xian Zhang (PhD student)

Contract Number: FA48690610032

AFOSR/AOARD Reference Number: AOARD-06-4008

AFOSR/AOARD Program Manager: Dr. Tae-Woo Park

Period of Performance: 15 April 2006 - 14 April 2007

Submission Date: 11 June 2007

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **29 JUN 2007**	2. REPORT TYPE **FInal**	3. DATES COVERED **15-04-2006 to 14-04-2007**
4. TITLE AND SUBTITLE **Timed Formalisms for Plan Ontology and Processes**		5a. CONTRACT NUMBER **FA48690610032**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Jin Song Dong**		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **National University of Singapore,3 Science Drive 2,Singapore 117543,Singapore,sp,117543**		8. PERFORMING ORGANIZATION REPORT NUMBER **N/A**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **AOARD, UNIT 45002, APO, AP, 96337-5002**		10. SPONSOR/MONITOR'S ACRONYM(S) **AOARD**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) **AOARD-064008**

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In the context of military planning, recent research effort focuses more on specifying the static features and relations in the plan ontology. However, military plans, e.g. air campaign planning, have additional operations and timing requirements. In this project, the PI proposed an appropriate real-timed formalism, Timed CSP with some possible extensions, to model the plans. In this approach, Timed CSP is used to model the processes of the plans. The additional critical system requirements are then captured by the extensions. An associated prototype tool, HORAE, is developed for modeling, and reasoning the military plans

15. SUBJECT TERMS
Ontology, Dynamic Planning, Timed Formalism

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**20**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Objectives

To develop formal modeling techniques and tool for timed planning.

Status of effort

Initial version of timed formalism language based Timed CSP has been developed and a prototype tool has been completed.

Abstract

In the context of military planning, recent research effort more focuses on specifying the static features and relations in the plan ontology. However, military plans, e.g., air campaign planning, have additional operations and timing requirements. In this project, we propose an appropriate real-timed formalism, Timed CSP with some possible extensions, to model the plans. In this approach, Timed CSP is used to model the processes of the plans. The additional critical system requirements are then captured by the extensions. An associated prototype tool, HORAE, is developed for modelling, and reasoning the military plans.

Personnel Supported

Dr. Sun Jun and Ms. Zhang Xian

Publications

J. S. Dong, P. Hao, J. Sun and X. Zhang, em A Reasoning Method for Timed CSP based on Constraint Solving. 8th International Conference on Formal Engineering Methods (ICFEM'06), pages 342-359, Springer, LNCS, November 2006.

Interactions

Participation/presentations at meetings, conferences, etc

paper presentation: 8th International Conference on Formal Engineering Methods (ICFEM'06) and
tools presentation: 14th International Symposium on Formal Methods (FM'06).

Potential technology application

The modeling techniques and tool can be potentially used as a plug-in by military planning tools.

New

List discoveries, inventions, or patent disclosures

Developed the first verification tool for Timed CSP (even though Timed CSP was proposed 20 years ago).

Completed the DD Form 882

See attached.

Honors/Awards

PI (J S Dong) was elected as a Visiting Fellow at Kellogg College, Oxford University UK (2006). Researcher (J. Sun) was awarded the prestigious Lee Kuan Yew Postdoctoral Fellowship (2007)

Archival Documentation

See the following detalied technical report.

Software and/or Hardware

The HORAE Beta-1.0 is made public: `http://comp.nus.edu.sg/~zhangxi5/horae`

Technical Report

Dr. Jin Song Dong (PI), Dr. Jun Sun (Postdoc), Xian Zhang (PhD student)

School of Computing,
National University of Singapore
{dongjs,sunj,zhangxi5}@comp.nus.edu.sg

1 Introduction

Military systems are often safety critical. Errors in military systems may even cause our own military casualties and civilian life. Recent research effort on military planning is more focusing on specifying the static features and relations in the plan ontology [10, 16, 3]. However, military plans, i.e., air campaign planning, have additional operations and timing requirements. For example, in the context of a plan that usually consists of various sub-plans, we may need to specify such scenarios: plan P should be finished within T time units; if sub-plan A cannot be completed at the time T, then stop A and switch to sub-plan B; during the execution of the sub-plan A, if an emergent event E happens within time T, then stop A and switch to sub-plan B. It is desirable to not only formally capture those scenarios, which are common to many military plans, but also perform some *timed plan feasibility analysis*, i.e., checking whether it is possible for a plan to fulfil some requirements.

In this project, we investigate the suitability of various existing real-timed formalisms that can precisely model plans. Timed CSP [13], an extension of CSP [5] by introducing a capability to quantify temporal aspects of sequencing and synchronization, is chosen as the modelling language. Timed CSP has been widely accepted and applied to a wide range of systems, including communication protocols, embedded systems, etc [14]. Timed CSP is a powerful language to model real-time reactive system, but we still need some extra critical constraints on the system which the traditional Timed CSP language is not capable to model, e.g., the separation time between two events, the duration of a sub-plan, and etc. We extend the Timed CSP with some possible features to complement the language in precisely modelling the timed military plans. An associated tool is needed to support the new timed plan formalism. Constraint Logic Programming (CLP [7]) is designed for mechanized proving based on constraint solving. CLP has been successfully applied to model programs and transition systems for the purpose of verification [9, 4], showing that their approach outperforms the well-know state-of-art systems with higher efficiency. In this work, we also develop a tool, HORAE, which uses the CLP($\mathbb{R}$) system [8] as the underlying reasoning engine and is implemented in JAVA. HORAE is capable to specify and timed plan feasibility analyze of the military plans modelled in this timed plan formalism.

The remainder of this report is organized as follows. Section 2 briefly introduces Timed CSP and Constraint Logic Programming. Section3 illustrates the reasoning method of Timed CSP in CLP. We mainly focus on the encoding of operational semantics of Timed CSP. Section 4 presents the extensions of the transitional Timed CSP with timed planning features and also how CLP handles these features. Section 5 concludes the report.

2 Background

2.1 Timed CSP

Timed CSP [2] extends the well-known CSP (Communicating Sequential Processes) notation of Hoare with timing primitives. As indicated by its name, CSP is an *event* based notation primarily aimed at describing the sequencing of behaviour within a process and the synchronisation of behaviour (or *communication*) between processes. Timed CSP extends CSP by introducing a capability to consider temporal aspects of sequencing and synchronisation.

CSP adopts a symmetric view of process and environment. Events represent a co-operative synchronisation between process and environment. Both process and environment may control the behaviour of the other by *enabling* or *refusing* certain events or sequences of events.

Process Primitives The primary building blocks for Timed CSP processes are sequencing, parallel composition, and choice.

A process which may participate in event a then act according to process description P is written

$$a@t \to P(t).$$

The event a is initially enabled by the process and occurs as soon as it is also enabled by its environment. The event a is sometimes referred to as the *guard* of the process. The (optional) timing parameter, t, records the time (relative to the start of the process) at which the event a occurs and allows the subsequent behaviour, P, to depend on its value.

The second form of sequencing is process sequencing. A distinguished event $\checkmark$ is used to represent and detect process termination. The sequential composition of P and Q, written $P;\ Q$, acts as P until P terminates by communicating $\checkmark$ and then proceeds to act as Q. The termination signal is hidden from the process environment. The process which may only terminate is written Skip.

The parallel evolution of processes P and Q, synchronised on event set X is written

$$P \,|[\, X \,]|\, Q.$$

No event from X is enabled in $P \,|[\, X \,]|\, Q$ unless enabled jointly by both P and Q. Other events occur in either P or Q separately.

Diversity of behaviour is introduced through two choice operators. The external choice operator allows a process a choice of behaviour according to what events are enabled by its environment. The process

$$a \rightarrow P \;\square\; b \rightarrow Q$$

begins with both a and b enabled and performs the first to be enabled by its environment. Subsequent behaviour is determined by the event which actually occurred, P after a and Q after b respectively. External choice may also be written in an intentional form,

$$\square\; a : A \bullet P(a).$$

Internal choice represents variation in behaviour determined by the internal state of the process. The process

$$a \rightarrow P \sqcap b \rightarrow Q$$

may initially enable either a, or b, or both, as it wishes, but must act subsequently according to which event actually occurred. Again an intentional form is allowed.

An derived concept in CSP is the notion of *channel.* A channel is a collection of events of the form $c.n$: the prefix c is called the *channel name* and the collection of suffixes the allowed *values* of the channel. When an event $c.n$ occurs it is said that *the value n is communicated on channel c.* By convention, when the value of a communication on a channel is determined by the environment (external choice) it is called an *input* and when it is determined by the internal state of the process (internal choice) it is called an *output.* It is convenient to write $c?n : N \rightarrow P(n)$ to describe behaviour over a range of allowed inputs instead of the longer $\square\, n : N \bullet c.n \rightarrow P(n)$. Similarly the notation $c!n : N \rightarrow P(n)$ is used instead of $\bigsqcap n : N \bullet c.n \rightarrow P(n)$ to represent a range of outputs. Expressions of the form $c?n$ and $c!n$ do not represent events, the actual event is $c.n$ in both cases.

Recursion is used to given finite representations of non-terminating processes. The process expression

$$\mu\, P \bullet a?n \rightarrow b!f(n) \rightarrow P$$

describes a process which repeatedly inputs an integer on channel a, calculated some function f of the input, and then outputs the result on channel b. CSP specifications are typically written as a sequence of simultaneous equations in a finite collection of process variables. Such a specification $X == F(X)$ is implicitly taken to describe the solution to the vector recursion $\mu\, X \bullet F(X)$.

In general, the behaviour of a process at any point in time may be dependent on its internal state and this may conceivably take an infinite range of values. It is often not possible to provide a finite representation of a process without introducing some notation for representing this internal process state. The approach adopted by CSP is to allow a process definition to be intentionally parameterised by state variables. Thus a definition of the form

$$P_{n:N} \mathrel{\widehat{=}} Q(n)$$

represents a (possibly infinite) family of definitions, one for each possible value of n. It is important to note that there is no inherent notion of process state in CSP, but rather that this intentional form of expression is a convenient way to provide a finite representation of an infinite family of process descriptions.

Timing Primitives To the standard CSP process primitives, Timed CSP adds two time specific primitives, the delay and the timeout.

A process which allows no communications for period t then terminates is written $\textsc{Wait}\, t$. The process

$$\textsc{Wait}\, t;\ P$$

is used to represent P delayed by time t. A process $e \xrightarrow{t} P$ delays process P by t time units after engaging event e.

The timeout construct passes control to an exception handler if no event has occurred in the primary process by some deadline. The process

$$a \to P \rhd\{t\}\ Q$$

will pass control to Q if the a event has not occurred by time t, as measured from the invocation of the process.

Example 1 (Air Campaign Mission). We are planning an air attack mission to destroy the military force of a designated area where most of the military forces of the enemy are centralized. This mission is consists of three sub tasks. Task A, namely *Bomb*, is to destroy the radar of the enemy. Task B is the main attack, where a team of ground-attack aircrafts will fly to the specific area and give a fierce attack to the targeted military bases. If this task cannot be completed within 60 minutes, then it will switch to task C which is sending the air reinforce to the main air attack.

$$\begin{aligned}
&\mathit{Bomb} = \mathit{assemble} \to \mathit{equipment} \to \mathit{advance} \xrightarrow{5} \mathit{lunchMissile} \xrightarrow{10} \mathit{retreat} \to \mathit{SKIP}\\
&\mathit{AirAttack} = \mathit{assemble} \to \mathit{equipment} \to \mathit{advance} \xrightarrow{5} \mathit{arrive} \xrightarrow{5} \mathit{attack} \to \mathit{SKIP};\\
&\qquad\qquad (\mathit{victory} \to \mathit{retreat} \to \mathit{SKIP}) \sqcap (\mathit{lose} \to \mathit{retreat} \to \mathit{SKIP})\\
&\mathit{Reinforce} = \mathit{assemble} \to \mathit{equipment} \to \mathit{advance} \xrightarrow{5} \mathit{supporting\_attack} \to \mathit{SKIP};\\
&\qquad\qquad (\mathit{victory} \to \mathit{retreat} \to \mathit{SKIP}) \sqcap (\mathit{lose} \to \mathit{retreat} \to \mathit{SKIP})\\
&\mathit{Mission} = \mathit{Bomb};\ (\mathit{AirAttack} \bigtriangleup_{60} \mathit{Reinforce})
\end{aligned}$$

2.2 CLP Preliminaries

Constraint Logic Programming (CLP) began as a natural merger of two declarative paradigms: constraint solving and logic programming. This combination helps make CLP programs both expressive and flexible, and in some cases, more efficient than other kinds of programs. The CLP scheme defines a class of languages based upon the paradigm of rule-based constraint programming, where CLP($\mathcal{R}$) is an instance of this class. We present some preliminary definitions about CLP [7].

Example 2 (Factorial). The following is typical CLP program:

$$\begin{array}{l} fac(0,1). \\ fac(N, X_1 * N) :- N > 0, fac(N-1, X_1). \end{array}$$

A relation *fac(N, X)* is defined, where X is the factorial of N, denoted as $X = N!$. There are two atoms for the relation *fac(N, X)*, where the first atom is a *fact* and the second one is a *rule*.

The *universe of discourse* $\mathcal{D}$ of our CLP program is a set of terms, integers, and lists of integers. A *Constraint* is written using a language of functions and relations. They are used in two ways, in the basic programming language to describe expressions and conditions, and in user assertions, defined below. In this paper, we will not define the constraint language explicitly, but invent them on demand in accordance with our examples. Thus the terms of our CLP programs include the function symbols of the constraint language.

An *atom*, is as usual, of the form $p(\tilde{t})$, where p is a user defined predicate symbol and $\tilde{t}$ is a sequence of terms. A *rule* is of the form $A :- \tilde{B}, \Psi$ where the atom A is the *head* of the rule, and the sequence of atoms $\tilde{B}$ and the constraint Ψ constitute the *body* of the rule. A *goal* has exactly the same format as the body of the rule of the form $? - \tilde{B}, \Psi$. If $\tilde{B}$ is an empty sequence of atoms, we call this a (constrained) *fact*. All goals, rules and facts are terms. A *ground instance* of a constraint, atom and rule is defined in obvious way. A *ground instance* of a constraint is obtained by instantiating variables therein from $\mathcal{D}$. The *ground instances* of a goal G, written $[\![G]\!]$ is the set of ground atoms obtained by taking all the true ground instances of G and then assembling the ground atoms therein into a set. We write $G_1 \models G_2$ to mean that for all groundings θ of G_1 and G_2, each ground atom in $G_1\theta$ appears in $G_2\theta$.

Let $G = (B_1, ..., B_n, \Psi)$ and P denote a goal and program respectively. Let $R = A :- C_1, ..., C_m, \Psi_1$ denote a rule in P, written so as none of its variables appear in G. Let $A = B$, where A and B are atoms, be shorthand for equations between their corresponding arguments. A *reduct* of G using R is of the form

$$(B_1, ..., B_{i-1}, C_1, ..., C_m, B_{i+1}, ..., B_n, B_i = A \wedge \Psi \wedge \Psi_1)$$

provided $B_i = A \wedge \Psi \wedge \Psi_1$ is satisfiable. A *derivation sequence* is a possibly infinite sequence of goals $G_0, G_1, ...$ where $G_i, i > 0$ is a reduct of G_{i-1}. If there is a last goal G_n with no atoms, notationally $(\Box, \Psi)$ and called a *terminal goal*, we say that the derivation is a *successful* and that the *answer constraint* is Ψ. A derivation is ground if every reduction therein is ground.

Example 3 (Derivation). We calculate the 3! through the goal $? -\ \ fac(3,\ X)$. The following demonstrates a derivation sequence of the goal with three steps. The constraints in the last step which are the termination goal answer $X = 6$.

$$
\begin{array}{l}
N = 3,\ fac(N,\ X). \\
\qquad\qquad\qquad\qquad\Downarrow \\
N = 3, N > 0, N - 1 = N_1, X = N * X_1, fac(N_1, X_1). \\
\qquad\qquad\qquad\qquad\Downarrow \\
N = 3, N > 0, N - 1 = N_1, X = N * X_1, \\
N_1 > 0, N_1 - 1 = N_2, X1 = N_1 * X_2, fac(N_2, X_2). \\
\qquad\qquad\qquad\qquad\Downarrow \\
N = 3, N > 0, N - 1 = N_1, X = N * X_1, N_1 > 0, N_1 - 1 = N_2, \\
X1 = N_1 * X_2, N_2 > 0, N_2 - 1 = 0, X_2 = 1.
\end{array}
$$

3 Reasoning Method for Timed CSP

3.1 Timed CSP Semantics in CLP

This section is devoted to an encoding of the semantics of Timed CSP in CLP. The practical implication is that we may then use powerful constraint solver like CLP(R) [8] to do various proving over systems modelled using Timed CSP. Both the operational semantics and denotational semantics are encoded. The encoding of operational semantics serves most of our purposes. Nevertheless the encoding of the denotational semantics offers an alternative way of proving systems modelled in Timed CSP as well as the correctness of the encoding itself.

The very initial step of our work is the syntax encoding of Timed CSP process in CLP syntax, which can be automated easily by syntax rewriting. A relation of the form $proc(N, P)$ is used to present a process P with name N. For instance, Figure 1 is the syntax encoding of process *Military Mission Plan* in CLP, which is a timed interrupt process with name *mission*.

3.2 Operational Semantics

The operational semantics of Timed CSP is precisely defined by Schneider [15] using two relations: an evolution relation and a timed event transition relation. It is straightforward to verify that our encoding conforms the two relations in [15].

A relation of the form *tos(P1,T1,E,P2,T2)* is used to denote the *t*imed *o*perational *s*emantics, by capturing both evolution relations and timed event transition relations. Informally speaking, *tos(P1,T1,E,P2,T2)* is true if the process *P1* may evolve to *P2* through either a timed transition, i.e., let *T2-T1* time units pass, or an event transition by engaging an abstract event instantly[1]. The relation *tos* defines a transition system interpretation of a Timed CSP process,

[1] Or both at the same time by engaging an nontrivial action which takes time (necessary for only extensions to Timed CSP like TCOZ [11] where E could be a complicated computation)

```
proc(mission, sequence(P1, P2)) : −proc(bomb, P1), proc(attack, P2).
proc(attack, tinterrupt(P1, P2, 60)) : −proc(mainattack, P1), proc(reinforce, P2).
proc(bomb, eventprefix(assemble, eventprefix(equipment, P))) : −proc(bomb1, P).
proc(bomb1, delay(advance, delay(lunchMissile, eventprefix(retreat, skip), 10), 5)).
proc(mainattack, sequential(P1, P2)) : −proc(ma1, P1), proc(ma2, P2).
proc(ma1, eventprefix(assemble, eventprefix(equipment, P))) : −proc(ma11, P).
proc(ma11, delay(advance, delay(arrive, eventprefix(attack, skip), 5), 5).
proc(ma2, intchoice(P1, P2)) : −proc(ma21, P1), proc(ma22, P2).
proc(ma21, eventprefix(vicotry, eventprefix(retreat, skip))).
proc(ma22, eventprefix(lose, eventprefix(retreat, skip))).
proc(reinforce, sequential(P1, P2)) : −proc(rf1, P1), proc(rf2, P2).
proc(rf1, eventprefix(assemble, eventprefix(equipment, P))) : −proc(rf11, P).
proc(rf11, delay(advance, eventprefix(supportingattack, skip), 5)).
proc(rf2, intchoice(P1, P2)) : −proc(rf21, P1), proc(rf22, P2).
proc(rf21, eventprefix(vicotry, eventprefix(retreat, skip))).
proc(rf22, eventprefix(lose, eventprefix(retreat, skip))).
```

Fig. 1. Military Mission Planning in CLP

where the state is identified by the combination of the process expression and the time variable. Using tabling mechanism offered in some of the constraint solvers like CLP($\mathcal{R}$) [8] or XSB [17], the termination of the derivation sequence based on relation *tos* depends on the finiteness of the reachable process expressions from the initial one. Therefore, if a process is irregular (i.e. its trace is irregular as in automata theory), proving of goals which need to explore all reachable process expressions is not feasible. However, even for irregular processes, interesting proving like existence of a trace is still possible.

We define the *tos* relation in terms of each and every operator of Timed CSP. For the moment, we assume the process is not parameterized and we shall handle parameterized processes uniformly in Section 4. For instance, the primitive process expressions in Timed CSP are defined through the following clauses:

```
tos(stop, T1, [], stop, T2) : −D >= 0, T2 = T1 + D.
tos(skip, T, [termination], stop, T).
tos(skip, T1, [], skip, T2) : −D >= 0, T2 = T1 + D.
tos(run, T, [_], run, T).
tos(run, T1, [], run, T2) : −D >= 0, T2 = T1 + D.
```

The only transition for process Stop is time elapsing. Process Skip may choose to wait some time before engaging event *termination* which is our choice of representation for event ✓ in CLP. Process Run may either let time pass or engage any event. In the following, we show how hierarchical operators are encoded in CLP using the alphabetized parallel composition operator as an example.

In the operational semantics, the event transition and evolution transition associated with the alphabetized parallel composition operator the alphabetized parallel composition operator $P_1 \; {}_X||_Y \; P_2$ are illustrated as the following [15]:

$$\frac{P_1 \xrightarrow{e} P_1'}{P_1 \; {}_X||_Y \; P_2 \xrightarrow{e} P_1' \; {}_X||_Y \; P_2} [\; e \in X \cup \{\tau\} \setminus Y \;]$$

$$\frac{P_2 \xrightarrow{e} P_2'}{P_1 \; {}_X||_Y \; P_2 \xrightarrow{e} P_1 \; {}_X||_Y \; P_2'} [\; e \in Y \cup \{\tau\} \setminus X \;]$$

$$\frac{P_1 \xrightarrow{e} P_1', P_2 \xrightarrow{e} P_2'}{P_1 \; {}_X||_Y \; P_2 \xrightarrow{e} P_1' \; {}_X||_Y \; P_2'} [\; e \in X \cap Y \;]$$

$$\frac{P_1 \overset{d}{\rightsquigarrow} P_1', P_2 \overset{d}{\rightsquigarrow} P_2'}{P_1 \; {}_X||_Y \; P_2 \overset{d}{\rightsquigarrow} P_1' \; {}_X||_Y \; P_2'}$$

The $\rightarrow$ represents an event transition, whereas $\rightsquigarrow$ represents an evolution transition. The rules associated with the alphabetized parallel composition operator are as the following:

$$\begin{array}{l}
tos(para(P1, P2, X, Y), T, [E], para(P3, P2, X, Y), T) \\
\quad :-tos(P1, T, [E], P3, T), member(E, X), not(member(E, Y)). \\
tos(para(P1, P2, X, Y), T, [E], para(P1, P4, X, Y), T) \\
\quad :-tos(P2, T, [E], P4, T), member(E, Y), not(member(E, X)). \\
tos(para(P1, P2, X, Y), T, [E], para(P3, P4, X, Y), T) \\
\quad :-tos(P1, T, E, P3, T), tos(P2, T, E, P4, T), \\
\qquad member(E, X), member(E, Y). \\
tos(para(P1, P2, X, Y), T1, [], para(P3, P4, X, Y), T1 + D) \\
\quad :-tos(P1, T1, [], P3, T1 + D), tos(P2, T1, [], P4, T1 + D).
\end{array}$$

The first two rules state that either of the components may engage an event as long as the event is not shared. The third rule states that a shared event can only be engaged simultaneously by both components. The last expresses that the composition may allow time elapsing as long as both the components do. Other parallel composition operation, like $|[\, X \,]|$ and $|||$, can be defined as special cases of the alphabetized parallel composition operator straightforwardly. There is a clear one-to-one correspondence between our rules and the operators which are partly illustrated in Appendix A and fully at our website[2]. Therefore, the soundness of the encoding can be proved by showing there is a bi-simulation relationship [12] between the transition system interpretation defined in [15] and ours, and the bi-simulation relationship can be proved easily via a structural induction.

[2] `http://comp.nus.edu.sg/~zhangxi5/horae`

For simplicity, we do restrict the form of recursion to $\mu X \bullet P(X)$, which means mutual recursion through process referencing has to be transformed before hand. The following clauses illustrate how recursion is handled, where N is the recursion point, i.e., X in $\mu X \bullet P(X)$) and P is the process expression, i.e., $P(X)$.

$tos(recursion([N,P],P1),T,[E],recursion([N,P],P2),T)$
$\quad :-not(P1 == N), tos(P1,T,[E],P2,T).$
$tos(recursion([N,P],P1),T1,[],recursion([N,P],P2),T1+D)$
$\quad :-D > 0, tos(P1,T1,[],P2,T1+D).$
$tos(recursion([N,P],N),T,[],recursion([N,P],P),T).$

3.3 Proving Properties of Timed CSP

This section is devoted to various proving we may perform over systems modelled using Timed CSP and then encoded in CLP. We implemented a prototype in one of the CLP solver, namely CLP($\mathcal{R}$). Any CLP assertion can be proved against a given real-time system. We also developed a number of shortcuts for easy querying and proving.

Using CLP, we may make explicit assertion which is neither just a safety assertion, nor just a liveness assertion. Yet it can be used for both purposes using a unique interpretation. In the following, we show how safety properties and liveness properties, like reachability, can be queried. We employ the concept of *coinductive tabling* with the purpose of obtain termination when dealing with recursions, which facilitates verifying safety and liveness properties based on traces. The detailed introduction of *coinductive tabling* can be found in [6].

Because Timed CSP is an event-based specification language, it is clearly useful to prove safety and liveness properties in terms of predicate concerning not only state variables but also events. A discussion on how to allow such temporal properties is presented in [1]. In order to explore the full state space, we define the following[3]:

$treachable(P,P,[],T1,T1).$
$treachable(P,Q,N,T1,T2) :-tos(P,T1,[tau],P1,T3), nottable(P1),$
$\qquad assert(table(P1)), treachable(P1,Q,N,T3,T2).$
$treachable(P,Q,[t(D) \mid N],T1,T2) :-tos(P,T1,[t(D)],P1,T3),$
$\qquad nottable(P1), assert(table(P1)),$
$\qquad treachable(P1,Q,N,T3,T2), notN = [t(\_) \mid_{]}\ .$
$treachable(P,Q,[E \mid N],T1,T2) :-tos(P,T1,[E],P1,T3),$
$\qquad not(E = t(\_);\ E == tau;\ E = reccall(\_)), nottable(P1),$
$\qquad assert(table(P1)), treachable(P1,Q,N,T3,T2).$

The relation *treachable(P, Q, N, T1, T2)* states that it is possible to reach the process expression Q at time *T2* from P at time T1, with trace N. where *tau* is the internal event τ and $t(\_)$ denotes the time elapsing. By using the tabling

[3] The possible state variables and local clocks are skipped for simplicity.

method, we dynamically record the process expressions that have been explored so as to avoid re-exploring them. In this regard, one kind of liveness property namely reachability is easily asserted using *treachable*.

An invariant property (a predicate over time variable and state variables and possible local clocks) is in general expressed as the assertion:

$$inv(P, T, Property) : -not(treachable(P, Q, \_, T, T1), not\ sat(Property)).$$

where *not sat(Property)* is a constraint indicating that the output from the previous atom not satisfying the user defined *Property*.

One safety property of special interest is deadlock-freeness. The following clauses are used to prove it.

$$\begin{aligned}&deadlock(P, T1, N) : -treachable(P, P1, N, T1, T2),\\ &\quad (tos(P1, T2, [t(\_)], Q1, T3) \rightarrow not(tos(Q1, T3, E, \_, \_), not\ E = [t(\_)]);\\ &\qquad\qquad not(tos(P1, T2, E, Q1, T3), notE = [t(\_)])),\\ &\quad printf(\text{"}deadlock\ at\ trace : \%\backslash n\text{"}, [N]).\end{aligned}$$

Basically, it states that a process P at time *T1* may result in deadlock if it can reach the process expression *P1* at time *T2* where no event transition is available neither at *T2* nor at any later moment. The last line outputs the deadlocked trace as a counterexample. Alternatively, we may present it as a result of the deadlock proving.

We allow trace-based properties (safety or liveness) that can be checked by exploring trace set partially. The retrieve of a trace is done by the predicate $superstep(P, N, Q)$, which finds a sequence of events through which process expression P evolves to Q:

$$\begin{aligned}&superstep(P, [\_], \_) : -not(tos(P, \_, \_, Q, \_), not\ table(Q)).\\ &superstep(P, [A \mid N], Q) : -tos(P, \_, M, P1, \_), not(M == [];\ M == [tau]),\\ &\quad M = [A], not\ table(P1), assert(table(P1)), superstep(P1, N, Q).\\ &superstep(P, N, Q) : -tos(P, \_, M, P1, \_), (M == [];\ M == [tau]),\\ &\quad not\ table(P1), assert(table(P1)), superstep(P1, N, Q).\end{aligned}$$

We may prove that some event will always eventually be ready to be engaged using the following rule: where rule $member(E, N)$ returns true if event E appears at least once in the event sequence N.

$$finally(P, E) : -not(superstep(P, N, \_), not\ member(E, N)).$$

Predicate $finally(P, E)$ captures the idea that there is no such trace without event E in this process P. In other words, this process will eventually go to event E. Another property based on traces would be identifying the relationship among events, e.g., event A can never happen before (after) event B in a trace or trace fragment.

Example 4 (Verification). For the military mission example, we can verity that it is not possible that at the end of this mission, the army is neither win or lose, by executing the following goal, which returns false.

$$\begin{aligned}&?-proc(mission, P), superstep(P, N, stop),\\ &\quad not\ (member(win, N);\ member(lose, N)).\end{aligned}$$

In reality, most processes are non-terminating, so it would not be possible to retrieve all possible traces of a process. However, by given a specific trace of a trace fragment, we are able to identify whether it is an event sequencing of a given process. For instance, the following clause is used to query if a sequence of event is a trace of the system, where P is a process expression and X is a sequence of events.

$$trace(P, X) : -superstep(P, X,_) .$$

In addition to proving pre-specified assertions, one distinguished feature of our approach is that implicit assertions may be proved. For example, we may identify the lower or upper bound of a (time or data) variable, which is very useful for applications like worst or best case analysis of execution time.

$$\begin{aligned}&dur(P, Q, T1, T2) : -tos(P, T1, \_, Q, T2).\\ &dur(P, Q, T1, T2) : -tos(P, T1, \_, P1, T3), dur(P1, Q, T3, T2).\end{aligned}$$

We are able to compute the duration of the execution of one process P to its subsequent process Q by the above two rules, where T_1 is the starting time and T_2 is the ending time. By using the predicate *dur*, we are able to get identify the lower bound of some processes involving time. The process $\textsc{Wait}(2);\ a \xrightarrow{3} \textsc{Skip}$ should terminate in more than 5 time units, which can be identified by the following goal and expecting T≥5.

$$?-\ dur(sequential(wait(2), delay(a, skip, 3)), stop, 0, T).$$

3.4 Improvement of the Reasoning Engine

The execution time and state space of reasoning properties of a Timed CSP system mainly lie on the number of events of the system. Generally speaking, the more events in a system, the more state space and execution time it takes.

We make some abstraction on the existing Timed CSP system by eliminating some events according to the property we want to check, without affecting the semantics of the system related to this property. For the *deadlock* property, we can eliminate all the events which are not in the parallel interfaces and also not the control events (i.e., not the first event of the interrupt process).

$$\begin{aligned}&P1 = a \rightarrow b \xrightarrow{2} c \rightarrow d \rightarrow P1\\ &P2 = e \xrightarrow{1} (b \xrightarrow{2} c \rightarrow P2) \,\Box\, (c \rightarrow P2)\\ &P = P1\,|[\,b, c\,]|\,P2\end{aligned}$$

To check the property: $deadlock(P)$, we simply the process P to P' by eliminating event a,and e .

$$\begin{aligned}
&P1 = b \overset{2}{\rightarrow} c \rightarrow P1\\
&P2 = WAIT(1);\ (b \overset{2}{\rightarrow} c \rightarrow P2) \,\Box\, (c \rightarrow P2)\\
&P' = P1 \,|[\, b, c \,]|\, P2
\end{aligned}$$

¿From P' we can easily check that P is a deadlock free process.

4 Timed CSP with Timed Planning

As exemplified in the previous sections, Timed CSP captures a large set of behavior patterns in military planning. There are, however, relatively few design-independent planning requirements. For instance, a quick attack must finish in a few days, the number of loadings must be greater than the number of firings for any weapon, etc. In other words, there might be process independent system constraints which have to be enforced. Any successful planning must take these constraints into consideration. Thus, we extend the existing Timed CSP by adding a *where* clause to specify such constraints. Using the constraint solver, we can not only tell if planning is possible but also automatically generate a feasible plan, if there is any.

4.1 Syntax

Each process is extended with an optional *where* clause, which consists of a (first order) predicate over a predefined set of time variables. For instance, given a process P, the variable $P.\textsc{start}$ ($P.\textsc{end}$) denotes the exact starting (ending) time of the process. More specifically, $P.\textsc{start}$ captures the starting time of P when its first event is enabled or when the WAIT d process is enabled if P starts with the WAIT process. $P.\textsc{end}$ captures the ending time of P which is the same as the occurrent time of the termination event $\checkmark$. Naturally, $P.\textsc{end} \geq P.\textsc{start}$ all the time. Using the two variables, a *deadline* property (a task must be accomplished within certain time) is expressed as P **where** $P.\textsc{end} - P.\textsc{start} \leq d$ where d is constant. In other scenarios, there may be a requirement on some event to occur at some exact time, e.g., the marching of the troops must begin after half an hour and no later than one hour of the air force bombing. A time variable ENGAGE may be attached to an event to denote the exact time when e is engaged.

Example 5. The following example illustrates the use of the time variables,

$$\begin{aligned}
Task \,\widehat{=}\, & Bombing;\ march \rightarrow Sweeping\\
& \textbf{where}\ Bombing.\textsc{end} - Bombing.\textsc{start} < 172800\ \wedge\\
& Bombing.\textsc{end} + 3600 \geq march.\textsc{engage} \geq Bombing.\textsc{end} + 1800
\end{aligned}$$

The bombing finishes in two days and the marching must start half an hour after the bombing stops (so as to secure the ground before enemy returns). □

In addition, we use the variable $P.tr$ where P is an optional process to denote an arbitrary trace of process P. If P is missing, it is default as the process name on the left hand side.

Example 6. The following example illustrates a constraint using the variable tr. Assume that a gunboat is equipped with two canons. There are two soldiers, one in charge of loading shells (one of the canon at a time) and the other firing (one of the canon at a time).

$$
\begin{array}{ll}
\mathit{Loading} & \mathrel{\widehat{=}} \mathit{load}_1 \rightarrow \mathit{Loading} \mathrel{\square} \mathit{load}_2 \rightarrow \mathit{Loading} \\
\mathit{Firing} & \mathrel{\widehat{=}} \mathit{aim}_1 \rightarrow \mathit{fire}_1 \rightarrow \mathit{Firing} \mathrel{\square} \mathit{aim}_2 \rightarrow \mathit{fire}_2 \rightarrow \mathit{Firing} \\
\mathit{Gunboat} & \mathrel{\widehat{=}} \mathit{Loading} \parallel \mathit{Firing} \\
 & \quad \mathbf{where}\ \textsc{tr} \downarrow \mathit{load}_1 \geq \mathit{fire}_1 \wedge \mathit{tr} \downarrow \mathit{load}_2 \geq \mathit{fire}_2
\end{array}
$$

where $tr \downarrow e$ denotes the number of occurrences of event e in the trace tr. We remark that e must be in the alphabet of the process. □

In the following, we define the syntax of the *where* clause.

$$CP ::= P \ \ \mathit{where}\ \ \mathit{WherePred}$$

$$
\begin{array}{rl}
\mathit{WherePred} ::= & \mathit{WherePre} \wedge \mathit{WherePred} \\
| & \mathit{WherePre} \vee \mathit{WherePred} \\
| & \mathit{WherePre} \Leftrightarrow \mathit{WherePred} \\
| & \mathit{WherePre} \Rightarrow \mathit{WherePred} \\
| & \neg\ \mathit{WherePre} \mid \mathit{true} \mid \mathit{false} \\
| & \mathit{WhereExpr}
\end{array}
$$

$$
\begin{array}{rll}
\mathit{WhereExpr} ::= & [\mathit{Name}.]\textsc{start} & \text{– start time of a process} \\
| & [\mathit{Name}.]\textsc{end} & \text{– end time of a process} \\
| & [\mathit{Name}.]\textsc{engage}_i & \text{– the } i_{th} \text{ engage time of e} \\
| & [\mathit{Name}.]\textsc{tr} & \text{– trace of P} \\
| & \mathit{WhereExpr}\ \ \mathit{Infix}\ \ \mathit{WhereExp} & \\
| & \mathit{Prefix}\ \ \mathit{Exp} \mid \mathit{Name} \mid (\mathit{WhereExp}) &
\end{array}
$$

This syntax does not define the details of the $\mathit{Infix}, \mathit{Prefix}, \mathit{PostFix}, \mathit{Name}$. A name is a sequence of characters starting an alphabet. Typical infix operators are { +, -, >, <} or $tr \downarrow e$ which is the number of e in the trace. An example prefix operator is $\#tr$ which denotes the length of the trace. Notice that $P.\textsc{start} \leq e.\textsc{enable}_i \leq e.\textsc{engage}_i \leq P.\textsc{end}$). For each event e in process P, its enable time must be smaller than its engaged time, while the start time of P must be smaller than the enabled time of e and the end time of P must be greater than the engaged time of e.

Example 7. The following demonstrates that sing the basic terms and constraint defined above, common planning requirements can be specified rather easily.

$$T.\textsc{end} \leq d$$

Task T must finish before time d.

$$e.\text{ENGAGE}_j - e.\text{ENGAGE}_i \geq t1 \wedge j > i$$

Minimum separation between two appearances of e is $t1$.

$$e.\text{ENGAGE}_j - e.\text{ENGAGE}_i \leq t2 \wedge j = i + 1$$

Maximum separation between two appearances of e is $t2$. □

Because the *where* clause attached to a process definition is local to the process, only events, processes and variables visible to the process may appear in the clause. This is by no means a restriction, instead, it allows a modular specification of planning constraints. Semantically, the *where* clause restricts the set of traces allowed by the process. Informally, the process serves as a possible implementation of the plan, whereas the *where* clause is the high-level requirements (that must be guaranteed for any implementation of the plan). For instance, is a local constraint on P which would only affect the behaviors of P. If P is a sub-process of Q, the rest processes of Q are affected by the constraint if and only if there are synchronization events between P and the rest of Q. Otherwise the other processes in Q would not be affected even though they also contain the events appear in the constraint. In contrast, all sub-processes of Q must satisfy the constraint on Q. In other words, P is constrained by its local constraint and the constraints of Q.

Example 8. The *Military Mission Planning* example defined in 1, some feasible time planning requirements can be added. For example, the *Bomb* task it a rapid attack, which should be finished within 30 minutes. the main attack fighters must arrive at the destination area in 10 minutes. Whenever the reinforce army receives the command from the headquarter to go to the war, it should take at most 15 minutes for its equipping and march till they start to attack. Moreover, the whole mission must be finished within 150 minutes.

$$\begin{aligned}
&Bomb = assemble \rightarrow equipment \rightarrow advance \xrightarrow{5} lunchMissile \xrightarrow{10} retreat \rightarrow SKIP\\
&\qquad \text{WHERE } Bomb.\text{END} - Bomb.\text{START} \leqslant 30\\
&AirAttack = assemble \rightarrow equipment \rightarrow advance \xrightarrow{5} arrive \xrightarrow{5} attack \rightarrow SKIP;\\
&\qquad (victory \rightarrow retreat \rightarrow SKIP) \sqcap (lose \rightarrow retreat \rightarrow SKIP)\\
&\qquad \textbf{where } arrive.\text{ENGAGE} - advance.\text{ENGAGE} \leqslant 10\\
&Reinforce = assemble \rightarrow equipment \rightarrow advance \xrightarrow{5} supporting\_attack \rightarrow SKIP;\\
&\qquad (victory \rightarrow retreat \rightarrow SKIP) \sqcap (lose \rightarrow retreat \rightarrow SKIP)\\
&\qquad \text{WHERE } supporting\_attack.\text{ENGAGE} - assemble.\text{ENGAGE} \leqslant 15\\
&Mission = Bomb;\ (AirAttack \bigtriangleup_{60} Reinforce)\\
&\qquad \textbf{where } Mission.\text{END} - Mission.\text{START} \leqslant 150
\end{aligned}$$

4.2 Timed Planning in CLP(R)

A military plan documented using the extended Timed CSP might not be feasible. For instance, $P \mathrel{\widehat{=}} load \xrightarrow{3} aim \xrightarrow{2} fire \rightarrow P$ *where* $fire.\text{ENGAGE}_i -$

$load.\textsc{engage}_i < 4$ is an inconsistent plan. This boils down to the critical question that how effective we can verify if a plan is feasible. In this section, we show that using CLP(R), we can not only do that but also automatically synthesize a schedule of the plan, if it is feasible.

After modelling the extended Timed CSP processes in CLP(R), the prerequisite is to check feasibility of this system before the simulation or reasoning or scheduling of this system. It is possible that there is a conflict among its *where* clause and its sub processes's where *clauses*. To perform this task, the conjunction of the *where* clauses of each process and its sub processes is checked.

Example 9. We plan a mission which is to fire a target. It contains two sub missions, namely, the *Supply* and *Fire*, which are defined as follows.

$$Supply \mathrel{\widehat{=}} fetch \xrightarrow{1} boxup \xrightarrow{1} carry \xrightarrow{2} unload \rightarrow SKIP$$
$$Fire \mathrel{\widehat{=}} load \xrightarrow{1} aim \xrightarrow{2} fire \rightarrow SKIP \textbf{ where } fire.\textsc{engage} - load.\textsc{engage} > 12$$
$$Mission \mathrel{\widehat{=}} Supply;\ Fire \textbf{ where } Mission.\textsc{end} - Mission.\textsc{start} < 10$$

If we look at each process and its *where* clause separately, they are all feasible. Let's check out the conjunction of these clauses in CLP(R) which produces a conjunction of a set of constraints:

$$\begin{aligned}
&Fire.\textsc{start} \leqslant Fire.\textsc{end} \wedge \\
&Fire.\textsc{start} \leqslant Fire.fire.\textsc{engage} \wedge \\
&Fire.fire.\textsc{engage} \leqslant Fire.\textsc{end} \wedge \\
&Fire.fire.\textsc{engage} - Fire.load.\textsc{engage} > 12 \wedge \\
&Fire.\textsc{start} \leqslant Fire.load.\textsc{engage} \wedge \\
&Fire.load.\textsc{engage} \leqslant Fire.\textsc{end} \wedge \\
&Mission.\textsc{start} \leqslant Fire.\textsc{start} \wedge \\
&Fire.\textsc{end} \leqslant Mission.\textsc{end} \wedge \\
&Mission.\textsc{end} - Mission.\textsc{start} < 10
\end{aligned}$$

The conjunction of the constraints returns false. Therefore this *Mission* process is not a feasible plan.

5 Conclusion and Further Work

In this project, we proposed a new timed planning formalism to model timed military plans with critical timing requirements. This new timed formalism is an extension to Timed CSP with timed planning features. The extensions (in *where* clause) are easily captured semantically in CLP, which is the underlying reasoning language for this formalism. In this project, we also built a tool to reason and analyze this new formalism, which to our knowledge, is the first mechanized reasoning support for Timed CSP. This tool uses CLP as underlying reasoning engine and is implemented in JAVA.

Our further work will include the enhencement of the tool to handle timed fairness issues and other advanced real-time concurrent features. If it is possible, we would like to build this tool as plug-in feature in existing Millitary planning tool (esp. Air Campaign Planning Tools).

Acknowledgements

We would like to thank Dr. Terence Lyons, Dr. Hiroshi Motoda, Dr. Bill Nace and Dr. Tae-Woo Park of AOARD/AFRL for many helpful suggestions and support. We are grateful to have received Dr. Hiroshi Motoda's insightful comments on the early draft version of this report. We appreciate suggestions on the future research issues from Dr. Nace during his visit in May 2007. We would like to thank Dr. Paul B. Losiewicz of AFOSR/EOARD for suggesting our work to AOARD office.

References

1. S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/Event-based Software Model Checking. In *Proceeding of Integrate Formal Methods 2004*, pages 128–147, 2004.
2. J. Davies and S. Schneider. A brief history of Timed CSP. *Theoretical Computer Science*, 138, 1995.
3. J. S. Dong, C. H .Lee, Y. F. Li, and H. Wang. Verifying DAML+OIL and Beyond in Z/EVES. In *The 26th International Conference on Software Engineering (ICSE'04)*. ACM/IEEE Press, May 2004.
4. G.l Gupta and E. Pontelli. A Constraint-based Approach for Specification and Verification of Real-time Systems. In *IEEE Real-Time Systems Symposium*, pages 230–239, 1997.
5. C.A.R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice-Hall, 1985.
6. A. Santosa J. Jaffar and R. Voicu. Modeling Systems in CLP with Coinductive Tabling. In *International Conference on Logic Programming*, 2005.
7. J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
8. J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The CLP(R) Language and System. *ACM Trans. Program. Lang. Syst.*, 14(3):339–395, 1992.
9. J. Jaffar, A. E. Santosa, and R. Voicu. A CLP Proof Method for Timed Automata. In *Real-Time Systems Symposium*, pages 175–186, 2004.
10. C. H .Lee, H. B .Lee, and G. W .NG. Plan Ontology and It's Application. In *Fusion'04*, 2004.
11. B. P. Mahony and J. S. Dong. Timed Communicating Object Z. *IEEE Trans. Software Eng.*, 26(2):150–177, 2000.
12. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Comput. Sci.* Springer-Verlag, 1980.
13. G. Reed and A. Roscoe. A timed model for communicating sequential processes. *Theoret. Comput. Sci.*, 58:249–261, 1988.
14. A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1997.
15. S. A. Schneider. An Operational Semantics for Timed CSP. In *Proceedings Chalmers Workshop on Concurrency, 1991*, pages 428–456. Report PMG-R63, Chalmers University of Technology and University of Göteborg, 1992.
16. M. Tallis, J. Kim, and Y.Gil. User studies of knowledge acquisition tools: Methodology and lessons learned. In *KAW'99*, 1999.
17. D. S. Warren. Programming with Tabling in XSB. In *PROCOMET '98: Proceedings of the IFIP TC2/WG2.2,2.3 International Conference on Programming Concepts and Methods*, pages 5–6, London, UK, 1998.

Appendix A: **Operational Semantics of Timed CSP in CLP(R)**

Operational Semantics of operations of Timed CSP defined in CLP(R), where *tau* indicates the internal event τ, and E is an arbitrary event.

$\mathit{tos}(\mathit{eventprefix}(E, P), T1, [], \mathit{eventprefix}(E, P), T1 + D) :- D > 0.$
$\mathit{tos}(\mathit{eventprefix}(E, P), T, [E], P, T).$
$\mathit{tos}(\mathit{prefixchoice}(X, P), T, [Y], P, T) :- \mathit{member}(Y, X).$
$\mathit{tos}(\mathit{prefixchoice}(\_, P), T1, [], P, T1 + D) :- D > 0.$
$\mathit{tos}(\mathit{timeout}(Q1, \_, \ ), T, [E], P, T) :- \mathit{tos}(Q1, T, [E], P, T).$
$\mathit{tos}(\mathit{timeout}(, Q2, D), T, [\mathit{tau}], Q2, T) :- D = 0.$
$\mathit{tos}(\mathit{timeout}(Q1, Q2, D), T, [\mathit{tau}], \mathit{timeout}(P, Q2, D), T)$
$\quad :- \mathit{tos}(Q1, T, [\mathit{tau}], P, T).$
$\mathit{tos}(\mathit{timeout}(Q1, Q2, D), T1, [], \mathit{timeout}(P, Q2, D - T), T1 + T)$
$\quad :- T > 0, T <= D, \mathit{tos}(Q1, T1, [], P, T1 + T).$
$\mathit{tos}(\mathit{wait}(D), T1, E, P, T2) :- \mathit{tos}(\mathit{timeout}(\mathit{stop}, \mathit{skip}, D), T1, E, P, T2).$
$\mathit{tos}(\mathit{extchoice}(P1,), T, [E], P3, T) :- \mathit{tos}(P1, T, [E], P3, T).$
$\mathit{tos}(\mathit{extchoice}(, P2), T, [E], P4, T) :- \mathit{tos}(P2, T, [E], P4, T).$
$\mathit{tos}(\mathit{extchoice}(P1, P2), T, [\mathit{tau}], \mathit{extchoice}(P3, P2), T) :- \mathit{tos}(P1, T, [\mathit{tau}], P3, T).$
$\mathit{tos}(\mathit{extchoice}(P1, P2), T, [\mathit{tau}], \mathit{extchoice}(P1, P4), T) :- \mathit{tos}(P2, T, [\mathit{tau}], P4, T).$
$\mathit{tos}(\mathit{extchoice}(P1, P2), T1, [], \mathit{extchoice}(P3, P4), T2)$
$\quad :- T2 > T1, \mathit{tos}(P1, T1, [], P3, T2), \mathit{tos}(P2, T1, [], P4, T2).$
$\mathit{tos}(\mathit{interleave}(P1, P2), T, E, \mathit{interleave}(P3, P2), T)$
$\quad :- \mathit{tos}(P1, T, E, P3, T), (E == [];\ E == [\mathit{tau}]).$
$\mathit{tos}(\mathit{interleave}(P1, P2), T, E, \mathit{interleave}(P1, P4), T)$
$\quad :- \mathit{tos}(P2, T, E, P4, T), (E == [];\ E == [\mathit{tau}]).$
$\mathit{tos}(\mathit{interleave}(P1, P2), T, [E], \mathit{interleave}(P3, P2), T) :- \mathit{tos}(P1, T, [E], P3, T).$
$\mathit{tos}(\mathit{interleave}(P1, P2), T, [E], \mathit{interleave}(P1, P3), T) :- \mathit{tos}(P2, T, [E], P3, T).$
$\mathit{tos}(\mathit{interleave}(P1, P2), T1, [], \mathit{interleave}(P3, P4), T1 + D)$
$\quad :- D > 0, \mathit{tos}(P1, T1, [], P3, T1 + D), \mathit{tos}(P2, T1, [], P4, T1 + D).$
$\mathit{tos}(\mathit{interleave}(P1, P2), T, [\mathit{termination}], \mathit{interleave}(P3, P4), T)$
$\quad :- \mathit{tos}(P1, T, [\mathit{termination}], P3, T), \mathit{tos}(P2, T, [\mathit{termination}], P4, T).$
$\mathit{tos}(\mathit{hiding}(P1, X), T, [\mathit{tau}], \mathit{hiding}(P2, X), T)$
$\quad :- \mathit{tos}(P1, T, [E], T, P2), \mathit{member}(E, X).$
$\mathit{tos}(\mathit{hiding}(P1, X), T, [E], \mathit{hiding}(P2, X), T)$
$\quad :- \mathit{tos}(P1, T, [E], P2, T), \mathit{not}(\mathit{member}(E, X)).$
$\mathit{tos}(\mathit{hiding}(P1, X), T1, [], \mathit{hiding}(P2, X), T1 + D)$
$\quad :- D > 0, \mathit{tos}(P1, T1, [], P2, T1 + D),$
$\qquad \mathit{not}(\mathit{member}(A, X), \mathit{tos}(P1, \_, [A], \_, \_)).$
$\mathit{tos}(\mathit{sequential}(P1, P2), T, [E], \mathit{sequential}(P3, P2), T)$
$\quad :- \mathit{tos}(P1, T, [E], P3, T), \mathit{not}(E = \mathit{termination}).$
$\mathit{tos}(\mathit{sequential}(P1, P2), T, [\mathit{termination}], P2, T) :- \mathit{tos}(P1, T, [\mathit{termination}],, T).$
$\mathit{tos}(\mathit{sequential}(P1, P2), T1, [], \mathit{sequential}(P3, P2), T1 + D)$
$\quad :- D > 0, \mathit{tos}(P1, T1, [], P3, T1 + D), \mathit{not}(\mathit{tos}(P1, \_, [\mathit{termination}], \_, \_)).$
$\mathit{tos}(\mathit{interrupt}(P1, P2), T, [E], \mathit{interrupt}(P3, P2), T) :- \mathit{tos}(P1, T, [E], P3, T).$
$\mathit{tos}(\mathit{interrupt}(\_, P2), T, [E], P3, T) :- \mathit{tos}(P2, T, [E], P3, T).$
$\mathit{tos}(\mathit{interrupt}(P1, P2), T1, [], \mathit{interrupt}(P3, P4), T1 + D)$
$\quad :- D > 0, \mathit{tos}(P1, T1, [], P3, T1 + D), \mathit{tos}(P2, T1, [], P4, T1 + D).$